2 OVERVIEW OF SYSTEM ENGINEERING

This section traces several key developments and lessons learned that led to today's championing of SE as a powerful approach to organizing and conducting complex programs, such as those found in the NAS. SE continues to evolve, with an emphasis on stronger commercial- and team-based engineering organizations, as well as organizations without technical products. Before World War II, architects and civil engineers were, in effect, system engineers who worked on large, primarily civil, engineering projects, including the Egyptian pyramids, Roman aqueducts, Hoover Dam, the Golden Gate Bridge, and the Empire State Building, while other architects worked on trains and large ships. However, "early" system engineers operated without any theory or science to support SE. Thus, they lacked defined and consistently applied processes or practices. During World War II, a program manager and chief engineer might oversee development of an aircraft program, while others managed key subsystems, such as propulsion, controls, structure, and support systems, leading to a lack of uniformity throughout the process.

Some additional SE elements, such as operations research and decision analysis, gained prominence during and after World War II. Today, with more complex requirements and systems, chief engineers use SE to develop requirements and to integrate the activities of the program teams.

SE began to evolve as a branch of engineering during the late 1950s. At this time—when both the race to space and the race to develop missiles equipped with nuclear warheads were considered absolutely essential for national survival—the military services and their civilian contractors were under extreme pressure to develop, test, and place in operation nuclear-tipped missiles and orbiting satellites. In this climate, the services and their contractors sought tools and techniques to improve system performance (mission success) and program management (technical performance, delivery schedule, and cost control). Engineering management evolved, standardizing the use of specifications, interface documents, design reviews, and formal configuration management. The advent of hybrid and digital computers permitted extensive simulation and evaluation of systems, subsystems, and components that facilitated accurate synthesis and tradeoff of system elements.

The lessons learned with development programs led to innovative practices in all phases of high-technology product development. A driving force for these innovations was attainment of high system reliability. Some examples of changes introduced during the period are:

- Parts traceability
- Materials and process control
- Change control
- Product accountability
- Formal interface control
- Requirements traceability

2.1 What Is System Engineering?

Beyond the definition used in the Introduction (Chapter 1), SE is an overarching process that trades off and integrates elements within a system's design to achieve the best overall product and/or capability known as a system. Although there are some important aspects of program management in SE, it is still much more of an engineering discipline than a management discipline. SE requires quantitative and qualitative decisionmaking involving tradeoffs, optimization, selection, and integration of the results from many engineering disciplines.

SE is iterative—it derives and defines requirements at each level of the system, beginning at the top (the NAS level) and propagating those requirements through a series of steps that eventually leads to a physical design at all levels (i.e., from the system to its parts). Iteration and design refinement lead successively to preliminary design, detail design, and final approved design. At each successive level, there are supporting lower-level design iterations that are necessary to gain confidence for decisions. During these iterations, many concept alternatives are postulated, analyzed, and evaluated in trade studies. These iterative activities result in a multi-tier set of requirements. These requirements form the basis for structured verification of performance. SE closely monitors all development activities and integrates the results to provide the best solution at all system levels.

2.2 What Is a System?

A system is an integrated set of constituent parts that are combined in an operational or support environment to accomplish a defined objective. These integrated parts include people, hardware, software, firmware, information, procedures, facilities, services, and other support facets. People from different disciplines and product areas have different perspectives on what makes up a system. For example, software engineers often refer to an integrated set of computer modules as a system. Electrical engineers might refer to a system as complex integrated circuits or an integrated set of electrical units. The FAA has an overarching system of systems called the NAS that includes, but is not limited to, all the airports; aircraft; people; procedures; airspace; communications, navigation, and surveillance/air traffic management systems; and facilities.

At times, it is difficult to agree on what comprises a system, as it depends entirely on the focus of those who define the objective or function of the system. For example, if the objective is to print input data, a printer may be defined as the system. However, another might consider the electricity required for the printer. Expanding the objective to processing input data and displaying the results yields a computer as the system. Further expansion of the objective to include a capability for computing nationwide or worldwide data and merging data/results into a database results in a computing network as the system, with the computer and printer(s) as subsystems of the system.

SE first defines the system at the top level, ensuring focus and optimization at that level, thus precluding narrow focus and suboptimization. It then proceeds to increasingly detailed lower levels until the system is completely decomposed to its basic elements. This hierarchy is described in the following paragraph.

2.2.1 System Hierarchy

A system may include hardware, software, firmware, people, information, techniques, facilities, services, and other support items. Figure 2.2-1 establishes a common reference for discussing

the hierarchy of a system/subsystem within the NAS. Each system item may have its own associated hierarchy. For example, the various software programs/components that may reside in a system have a commonly accepted hierarchy as depicted in Figure 2.2-2. Thus, Figure 2.2-2 is a subset of Figure 2.2-1 in that a system/subsystem may have multiple Computer Software Configuration Items (see definitions next page). The depths of this common hierarchy may be adjusted to fit the complexity of the system. Simple systems may have fewer levels in the hierarchy than complex systems and vice versa. Because there may be varying hierarchal models referenced in the realm of SE, it is important for those who define the objective or function of a given system/subsystem to also lay out the hierarchal levels of the system in order to define the system's scope.

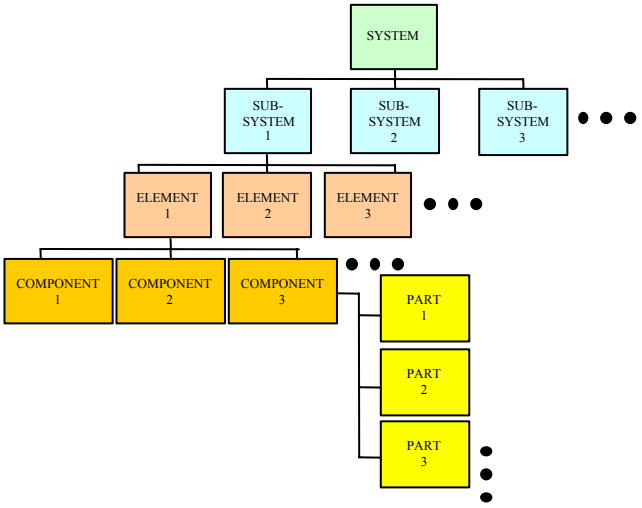

Figure 2.2-1. System Hierarchy

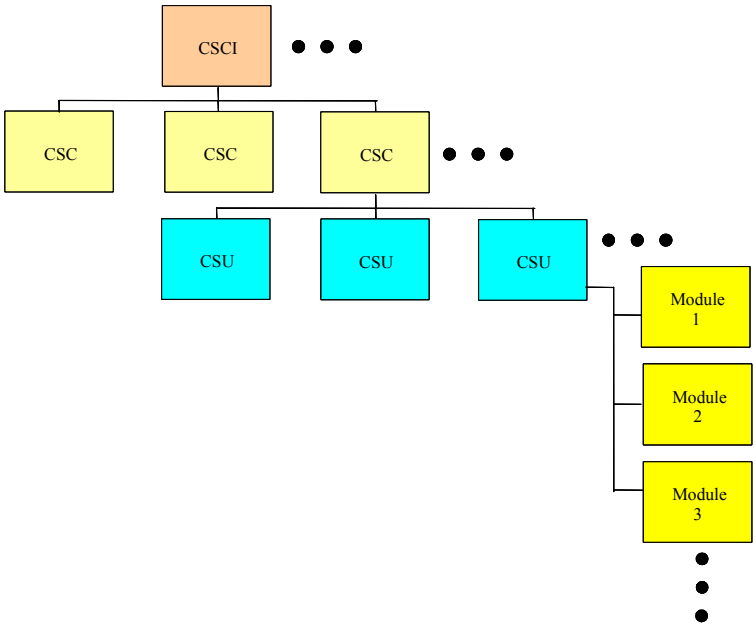

Figure 2.2-2. Common Software Hierarchy

Succeeding levels with the system/subsystem hierarchy are defined below:

- **System.** An integrated set of constituent parts that are combined in an operational or support environment to accomplish a defined objective. These parts include people, hardware, software, firmware, information, procedures, facilities, services, and other support facets.
- **Subsystem.** A system in and of itself (reference the system definition) contained within a higher-level system. The functionality of a subsystem contributes to the overall functionality of the higher-level system. The scope of a subsystem's functionality is less than the scope of functionality contained in the higher-level system.
- **Element.** An integrated set of components that comprise a defined part of a subsystem (e.g., the fuel injection element of the propulsion subsystem).
- **Component.** Composed of multiple parts; a clearly identified part of the product being designed or produced.
- **Part.** The lowest level of separately identifiable items within a system.
- **Software.** A combination of associated computer instructions and computer data definitions required to enable the computer hardware to perform computational or control functions.
- **Computer Software Configuration Item (CSCI).** An aggregation of software that is designed for configuration management and treated as a single entity in the Configuration Management process (Section 4.11).
- **Computer Software Component (CSC).** A functionally or logically distinct part of a CSCI, typically an aggregate of two or more software units.
- **Computer Software Unit.** An element specified in the design of a CSC that is separately testable or able to be compiled.
- **Module.** A program unit that is discrete and identifiable with respect to compiling, combining with other units, and loading.

2.3 Why Use System Engineering?

The most important reason to apply Systems Engineering is that it provides the context, discipline, and tools to adequately identify, define, and manage all system requirements in a balanced manner. It provides the disciplines required to produce a complete solution concept and system architecture. It also provides the discipline and tools to ensure that the resulting system meets all of the requirements that are feasible within specified constraints. No other engineering or management discipline explicitly provides this comprehensive context or results. The need for effective SE is most apparent with large, complex system developments, such as weapons and transportation systems. However, SE is also important in developing, producing, deploying, and supporting much smaller systems, such as cameras and printers. The growing complexity in development areas has increased the need for effective SE. For example, about 35 years ago in the semiconductor industry, a single chip was no more complex than a series of a few gates or, at most, a four-stage register. Today, Intel's Pentium processor is far more complex, which immensely expands the application horizon but demands far more sophisticated analysis and discipline in design.

The movement to concurrent engineering as the technique for performing engineering development is actually performing good SE. SE provides the technical planning and control

mechanisms to ensure that the activities/results of concurrent engineering meet overall system requirements.

A driving principle for SE is the teaming that often occurs during development programs. In this case, teaming is among several entities that may have different tools, analysis capabilities, and so on. SE principles defined in this manual may provide an improved ability to plan and control activities that require interaction and interfacing across boundaries.

The strongest argument for using the SE processes is that they increase the likelihood that needs may be fully and consistently met in the final product.